Appendix A
The lk Linker

This appendix describes the command-line interface to the lk linker. The lk linker is invoked when you specify the **–V lk_object** option on the **vcc** command line.

You must invoke the lk linker to link files produced by a version of vcc prior to Version 4.0 or files with a .obj extension. The default ld linker cannot link these files.

The lk linker, like the ld linker, is a linkage editor utility. It takes object modules as input and builds an executable image as output. The main purpose of both linkers is to resolve external references between modules in the program.

The output from the lk linker is a standard ULTRIX a.out object module.

A.1 The lk Command Line

The **lk** command line has the following format:

lk [*–option* [*option-args*]]... *file...*

***–option* [*option-args*]**
The **lk** command options are almost identical to the options accepted by the **ld** command, and both linkers process the options in the same fashion (that is, the individual options have the same effects). The only differences are that the **ld** command options **–A**, **–d**, and **–r** are not supported by the **lk** command, and the **lk** command option **–K** is not supported by the **ld** command.

NOTE

All **lk** command options can be specified with the **vcc** command.

Table A–1 describes the options for the **lk** command.

file
Files specified on the **lk** command line may be object files or library files. If a file is an object file (identified by the OMAGIC, NMAGIC, or ZMAGIC number at the appropriate place in the file), the linker processes the file's contents and concatenates the output to the program that it is building. Thus, the order in which object files are presented to the linker determines the order in which routines appear in the resulting program.

If the file is an archive library (identified by the presence of ARMAGIC at the appropriate place in the file), the linker searches the files in the library sequentially once for unresolved external symbols. If a resolution is found, that file is incorporated in the program being built and the search continues (possibly with new external symbols from the newly loaded file). If the library contains a ranlib-built dictionary file (_SYMDEF), the linker searches the dictionary iteratively to satisfy as many external references as possible instead of sequentially searching the archive.

Table A–1: Command Options Supported by the lk Linker

Command Option	**Description**
–A pathname	Puts incremental loading in effect. Linking is to be done so that the resulting object can be read into a program that is executing. The argument is the name of a file whose symbol table will be used to define additional symbols. Only newly linked material is entered into the text and data portions of a.out, but the new symbol table reflects every symbol defined before and after the incremental load. This argument must appear before any other object file in the argument list. You can use the **–T** option as well, which means that the newly linked segment will begin at the corresponding address (which must be a multiple of 1024). The default value is the old value of _end.
–D number	Sets the data segment length. Number is a number specifying the desired length of the data segment. The linker pads the data segment with 0 bytes to this length.
–e symbol	Sets the entry point. Symbol is the name of the entry point for the program. The default is location 0.
–H number	Sets the start of the data section. The linker adds the specified number to the end of the text address and causes the data section to start at that address.
–lx	Specifies a search library. The linker searches library /lib/libx.a (where x is the string specified as an argument to the **–l** option) for unresolved symbols. If this library does not exist, the linker searches /usr/lib/libx.a. If unresolved symbols remain, the linker tries /usr/local /lib/libx.a. The positioning of the **–l** option on the command line is significant: library searching occurs at the point in the link where the option occurs.
–K	Produces a full load map. The load map lists the files and modules included in the program, the allocation of psects to memory, and a cross-reference of all symbols (by name and by value). The map file is name.map, where name is the file name of the output file (as specified by the **–o** option). See Chapter 2 for a sample listing of a map file.
–M	Produces a short load map. **–M** is the same as **–K**, except that the symbolic cross-reference includes only those symbols that were referenced.
–N	The linker produces a file in OMAGIC format. The text portion of the file will be read/write and not shared.
–n	The linker produces a file in NMAGIC format. The text portion of the file will be read-only and shared among all users executing the file. The data segment starts at the first possible 1024-byte boundary following the end of the text segment.
–o file	Sets the output file name. The linker uses file as the name for the output file. The default is a.out.
–S	Removes some symbols. The linker does not include any symbols in the a.out file except for locals and globals.
–s	Removes all symbols and relocation information. The linker does not include any symbols or relocation information in the a.out file.

(continued on next page)

Table A–1 (Cont.): Command Options Supported by the lk Linker

Command Option	Description
–T number	Sets the text segment origin. The linker offsets the beginning of the text segment at the address given by the hexadecimal number specified as number. The default is 0.
–t	Traces. The linker writes the name of each file to standard output as it is processed.
–u symbol	Sets an undefined symbol. The linker adds the symbol name specified by symbol as an undefined symbol in the symbol table. This option is useful when loading entirely from libraries, since the symbol table is empty initially and an unresolved reference is needed to force the loading of the first routine.
–X	Removes symbols starting with L. The linker does not include symbols starting with L in the symbol table of the a.out file. Some compilers use such names for internally generated labels. The **–X** option provides a way to eliminate just these names.
–x	Removes local symbols. The linker includes only global symbols in the symbol table of the a.out file.
–Y [option]	Compile file for one of the following options: SYSTEM_FIVE, BSD, or POSIX.
–y symbol	Traces a symbol. The linker indicates each file that the specified symbol appears in and whether the file defines or references it. The **–y** option may occur many times to trace several symbols.
–z	The linker generates a.out in ZMAGIC format. The program is loaded on demand instead of being preloaded. This is the default format. The a.out header is 1024 bytes long. The text and data segments are padded with 0 bytes to the next higher 1024-byte boundary.

A.2 Linker Processing

The lk linker processes two different kinds of entities when building the executable program image: modules and program sections (psects).

A module is a unit of program compilation. There is usually one module per object file or library element. Some language translators, such as the as assembler, do not generate explicit modules. The linker assumes these translators to have one module consisting of the entire object file or library element.

A program section is a unit of virtual memory allocation. A program section specifies the attributes of the virtual memory to be allocated (for example, executable as opposed to data, read-only as opposed to read/write, absolute as opposed to relocatable, initialized as opposed to uninitialized). The linker uses the attributes of a program section to assign virtual memory addresses to the module contributions and to the symbols defined in the modules.

The psect also indicates the boundary alignment required for the segment of virtual memory. The data in an object module specifies, either explicitly or implicitly, the program section where the executable code and data in the module are to be allocated.

A.2.1 Program Section Attributes

This section describes each program section attribute and its effect on image processing by the lk linker. The attributes are grouped into mutually exclusive pairs as follows:

- Relocatable (REL) and Absolute (ABS)

  A relocatable program section is one that the linker can position in virtual memory according to the memory allocation strategy for the type of image being produced.

  The linker does not allocate virtual memory for an absolute program section. An absolute program section has no data or code, and appears as if it were based at virtual address 0. Absolute program sections are used primarily to define global symbols.

- Concatenated (CON) and Overlaid (OVR)

  These attributes govern how the linker allocates virtual memory to program section contributions from more than one module.

  If the program section has the concatenated attribute, the linker places each module's contribution in contiguous memory addresses. For example, if both modulea and moduleb contribute to program section psect1, and psect1 has the concatenated attribute, the linker allocates virtual memory for modulea's contribution and then allocates additional space for moduleb's contribution. Thus, the total size of a program section defined with the concatenated attribute is the sum of each module's contribution plus any padding allowed for individual alignments.

  If the program section has the overlaid attribute, the linker assigns each module's contribution the same base address, so that the contributions overlay each other. The total size of an overlaid program section is the size of the largest contribution. Any contribution to an overlaid program section can initialize the contents of the section. However, the final contents are determined by the last contributing module. Thus, the order in which you specify input modules is important.

  VAX FORTRAN common blocks and C external variables are implemented with overlaid program sections.

- Writeability (WRT and NOWRT)

  The writeability attribute determines whether the contents of the program section are protected against modification when you execute the program. Program sections with the nonwriteable attribute are allocated in the text section of the image. Program sections with the writeable attribute are allocated in the data or BSS sections, based on the modified attribute.

- Modified (MOD) and Unmodified (NOMOD)

  The modified attribute tells the linker whether any modules initialize the contents of the program section. Writeable, unmodified program sections are the only ones that the linker places in the BSS section of the image.

- Alignment

  The program section alignment attribute specifies the power-of-two boundary on which the linker is to align the program section code or data. The linker can align on any power of two in the range 0 (byte alignment) to 9 (512-byte boundary alignment).

- Local (LCL) and Global (GBL) Scope

If the global and overlaid attributes occur together, the program section is subject to special treatment from the linker for resolving references to .COMM symbols and to symbols with the same name as the program section. See Section A.2.3 for a description of how this is handled.

The following attributes are reserved for possible future implementation:

- Executability (EXE and NOEXE)
- Readability (RD and NORD)
- Position Independence (PIC and NOPIC)
- Shareability (SHR and NOSHR)
- User (USR) and Library (LIB)
- Protection (VEC and NOVEC)

A.2.2 Virtual Memory Allocation by the Linker

An ULTRIX executable program image file contains three virtual memory sections: a program text (executable code) section, an initialized data section, and a BSS (uninitialized data) section. The linker allocates virtual memory to the sections in that order: text, initialized data, BSS. The linker orders program sections in each image virtual memory section according to the program section attributes, as follows:

Image Section	Program Section Attributes			
text	NOWRT	EXE	NOVEC	—
	NOWRT	EXE	VEC	—
	NOWRT	NOEXE	NOVEC	—
	NOWRT	NOEXE	VEC	—
data	WRT	EXE	NOVEC	MOD
	WRT	EXE	VEC	MOD
	WRT	NOEXE	NOVEC	MOD
	WRT	NOEXE	VEC	MOD
BSS	WRT	—	—	NOMOD

Within an attribute group, the linker allocates program sections in alphabetical program section name order, with one exception: program section ULT$TEXT always occurs first in the text section.

Program sections with the global and overlaid attributes are handled in a special way. The linker constructs a new name by converting the program section name to lowercase and prefixing an underscore to it. If this name matches a symbol defined in a program section with different attributes (that is, other than GBL,OVR), the linker bases the global, overlaid program section at the virtual address given by the value of the symbol that was matched. This processing allows global, overlaid program sections to be initialized from a.out-format modules created by the as assembler, VAX C compiler, portable C compiler, portable Pascal compiler, and f77 compiler.

The virtual memory base address for the program image sections themselves depends on the type of image being generated as follows:

- Read/Write Text (OMAGIC format)

  The data section immediately follows the end of the text section, plus any padding specified by the **–H** command option. The BSS section immediately follows the end of the data section, plus any padding specified by the **–D** command option.

- Read-Only Text (NMAGIC format)

  The data section starts on the next page (1024-byte) boundary following the end of the text section, plus any padding from the **–H** command option. The BSS section immediately follows the end of the data section, plus any padding specified by the **–D** command option.

- Demand Loadable (ZMAGIC format)

  The data section starts on the next page (1024-byte) boundary following the end of the text section, plus any padding from the **–H** command option. The data segment, plus any padding specified by the **–D** command option, is padded to the next highest page boundary. Thus, the BSS section starts on the next page boundary following the end of the data section plus any padding specified by the **–D** command option. However, the linker will allocate program sections with the modified attribute starting at the end of the data section (plus **–D** value) to avoid wasting the space resulting from data segment roundup.

A.2.3 Special Processing for Modules Produced by the ld Linker

Modules in a.out(5) format, the format generated by the as assembler and processed by the ld linker, receive some special processing from the lk linker. These modules do not have any explicit module or program section declarations, and some symbols require special handling. The special rules for files and library elements with a.out(5) format are as follows:

- The linker considers the file or library element to be a single module with the same name as the file or library element.
- The module declares and makes contributions to three program sections. All program sections and contributions have alignment value 2 (longword alignment).

  1. The following program section receives the contents of the text section of the module:

     ```
     ULT$TEXT (PIC, USR, CON, REL, GBL, EXE, RD, NOWRT, NOVEC, MOD)
     ```

  2. The following program section receives the contents of the data section of the module:

     ```
     ULT$DATA (PIC, USR, CON, REL, GBL, NOEXE, RD, WRT, NOVEC, MOD)
     ```

  3. The following program section receives the contents of the BSS section of the module, and the symbols from the linker-generated module $$COMSYMS:

     ```
     ULT$COMM (PIC, USR, CON, REL, GBL, NOEXE, RD, WRT, NOVEC, NOMOD)
     ```

- The linker processes .COMM symbols from modules with a.out(5) format in a special way. These symbols represent C external variables, COMMON areas from the f77 compiler, and other such entities that the linker normally processes as program sections with the overlaid attribute.
  - If the name of a .COMM symbol matches a symbol that is defined elsewhere and is not a .COMM symbol, the .COMM symbol is treated as a reference to that symbol.
  - If the name of a .COMM symbol matches the name of a program section with the global and overlaid attributes (after the program section name is converted to lowercase and prefixed with an underscore), the linker treats the .COMM symbol as a reference to offset 0 in that program section.
  - If neither of the previous two conditions apply, the linker increases the size of program section $$COMSYMS. The size of the increase is the largest length attribute encountered for that .COMM symbol. The .COMM symbol is considered a definition of that name and its value is the offset of the contribution from the start of program section $$COMSYMS.